



C# Avançado

Métodos Anônimos e Expressões Lambda

Softblue
cursos online

Tópicos Abordados

- Métodos anônimos
 - Funcionamento
 - Omissão dos parâmetros
 - Acesso a variáveis externas
- Expressões lambda
 - Conceito
 - Sintaxe
 - O delegate *Predicate<T>*
 - Closures

Relembrando dos Events

```
class Pessoa
{
    public event EventHandler<AcordouEventArgs> Acordou;

    public void Acordar()
    {
        DateTime hora = DateTime.Now;

        if (Acordou != null)
        {
            Acordou(this, new AcordouEventArgs(hora));
        }
    }
}
```

```
static void Main()
{
    Pessoa p = new Pessoa();
    p.Acordou += PessoaAcordou;
}

static void PessoaAcordou(object sender, AcordouEventArgs args)
{
    //...
}
```

Método criado para ser chamado quando o evento ocorrer

Métodos Anônimos



- Métodos criados para serem chamados em respostas a eventos dificilmente são usados em outros contextos
 - O método existe apenas para tratar aquele evento específico
- Uma alternativa à criação destes métodos é criar *métodos anônimos*
 - São métodos que não têm um nome definido
- Métodos anônimos podem ser usados tanto com events como com delegates

Métodos Anônimos



- Sem o uso de métodos anônimos

```
static void Main()
{
    Pessoa p = new Pessoa();
    p.Acordou += PessoaAcordou;
}

static void PessoaAcordou(object sender, AcordouEventArgs args)
{
    //...
}
```

- Usando métodos anônimos

```
static void Main()
{
    Pessoa p = new Pessoa();
    p.Acordou += delegate(object sender, AcordouEventArgs args)
    {
        //...
    };
}
```

O método não tem nome, e é definido de forma *inline*

Não Uso dos Parâmetros



- Ao definir um método anônimo, não é preciso declarar os parâmetros se eles não forem usados pelo método

```
static void Main()
{
    Pessoa p = new Pessoa();
    p.Acordou += delegate
    {
        //...
    };
}
```

Os parâmetros não foram especificados

Variáveis em Métodos Anônimos



- Um método anônimo pode definir variáveis como qualquer método
- Ele pode acessar variáveis locais definidas pelo método que o define (*outer method*)

```
static void Main()
{
    Pessoa p = new Pessoa();
    int diaDoMes = 5;

    p.Acordou += delegate(object sender, AcordouEventArgs args)
    {
        Console.WriteLine(diaDoMes);
    };
}
```

Acesso à variável *diaDoMes* definida no *outer method*

Expressões Lambda



- O nome lambda vem do conceito matemático de *cálculo lambda*
- Em C#, uma expressão lambda é uma forma de simplificar o uso de métodos anônimos
- Qualquer método que recebe um delegate como parâmetro pode ser chamado usando uma expressão lambda

Exemplo de Método Anônimo



```
delegate bool Filter(int e);
```

```
static List<int> FilterList(List<int> list, Filter filterDelegate)
{
    List<int> newList = new List<int>();

    foreach (int e in list)
    {
        if (filterDelegate(e))
        {
            newList.Add(e);
        }
    }

    return newList;
}
```


Parâmetro do tipo delegate

```
List<int> list = new List<int>();
list.AddRange(new int[] { 8, 22, 10, 13, 6, 14 });

List<int> newList = FilterList(list, delegate(int e) {
    return e > 10;
});
```

Método anônimo

Exemplo de Expressão Lambda




```
List<int> newList = FilterList(list, delegate(int e) {
    return e > 10;
});
```

Método anônimo

```
List<int> newList = FilterList(list, e => e > 10);
```

Expressão lambda

Sintaxe de Expressões Lambda



- Uma expressão lambda é representada da seguinte forma

Parâmetros => Processamento

Operador lambda


- Exemplos de sintaxe

```
e => e > 10
(e) => (e > 10)
(int e) => e > 10;
(x, y) => x * y;
() => "abc"
```

```
e =>
{
    return e > 10;
}

(x, y) =>
{
    x = x + 1;
    y = y - 1;
    return x + y;
}
```

O Delegate *Predicate<T>*



- Um delegate bastante usado no C# é o *Predicate<T>*
 - T* especifica o tipo do parâmetro
 - O retorno é do tipo *bool*
- Bastante usado em conjunto com listas e arrays, para fazer filtro de dados

Exemplo de Uso do *Predicate<T>*

```

public class List<T>
{
    public List<T> FindAll(Predicate<T> match);
    ...
}

```

```

Predicate<int> p = x => (x % 2) == 0;
List<int> newList = list.FindAll(p);

```

```

List<int> newList = list.FindAll(x => (x % 2) == 0);

```

Retorna uma lista contendo os elementos que satisfazem o critério de busca

Closures

- Expressões lambda têm a capacidade de acessar variáveis definidas externamente
- Este recurso é denominado *closure*

```

static void Main()
{
    int mult = 2;
    Func<int, int> f = (x => x * mult);
    Console.WriteLine(f(5));
}

```

Variável definida fora da expressão lambda

A chamada retorna 10

Variáveis Externas

- O valor da variável externa usada pela expressão lambda é o valor do momento da invocação

```


static void Main()
{
    int mult = 2;
    Func<int, int> f = (x => x * mult);
    mult = 5;
    Console.WriteLine(f(5));
}

```

A expressão lambda tem acesso direto à variável, e não a uma cópia do valor

O resultado de $f(5)$ é 25

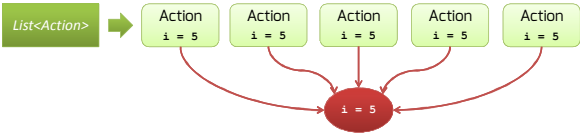
Closure em Loops




```
List<Action> actions = new List<Action>();
for (int i = 0; i < 5; i++)
{
    actions.Add(() => Console.WriteLine(i));
}

foreach (Action action in actions)
{
    action();
}
```

Resultado: 5, 5, 5, 5, 5



Closure em Loops



```
List<Action> actions = new List<Action>();
for (int i = 0; i < 5; i++)
{
    int j = i;
    actions.Add(() => Console.WriteLine(j));
}

foreach (Action action in actions)
{
    action();
}
```

Resultado: 0, 1, 2, 3, 4

Cada expressão lambda usa uma variável diferente